Documentation Technique Complète : API de Gestion de Zoo

- **Auteur** : [Votre Nom]
- **Date** : 31 juillet 2025
- **Projet** : Évaluation finale - API RESTful avec NestJS, TypeORM, et Auth0.

Partie I : Vision d'Ensemble et Architecture

1.1. Introduction

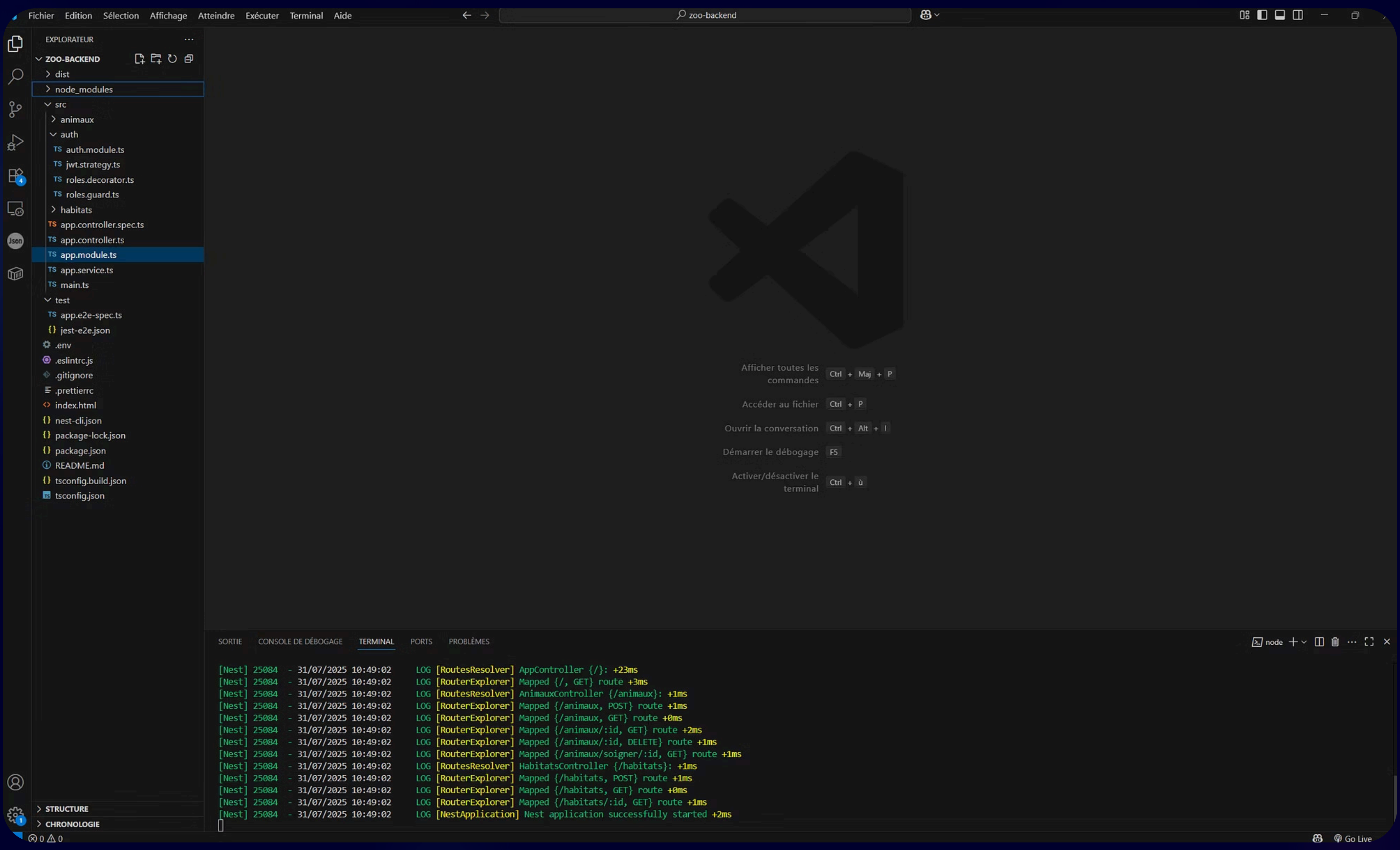
Ce projet consiste en la réalisation d'une API backend robuste pour la gestion d'un parc zoologique. L'application, développée avec le framework **NestJS**, offre une interface RESTful pour gérer des entités telles que les animaux et leurs habitats. La persistance des données est assurée par une base de données **PostgreSQL**, et la sécurité est gérée par **Auth0**, qui fournit un système d'authentification et d'autorisation basé sur les rôles (RBAC).

L'objectif était de démontrer la maîtrise des technologies backend modernes, la mise en place de flux de sécurité complexes, et la capacité à produire un projet complet, testable et bien documenté.

1.2. Architecture Logicielle

L'architecture du projet s'articule autour des principes de modularité et de séparation des préoccupations, qui sont au cœur de NestJS.

Légende : Vue d'ensemble de l'organisation modulaire du projet, montrant les dossiers src/animaux, src/habitats, et src/auth.



```
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RoutesResolver] AppController {/}: +23ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/, GET} route +3ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RoutesResolver] AnimauxController {/animaux}: +1ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/animaux, POST} route +1ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/animaux, GET} route +0ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/animaux/:id, GET} route +2ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/animaux/:id, DELETE} route +1ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/animaux/soigner/:id, GET} route +1ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RoutesResolver] HabitatsController {/habitats}: +1ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/habitats, POST} route +1ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/habitats, GET} route +0ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [RouterExplorer] Mapped {/habitats/:id, GET} route +1ms
[Nest] 25084  - 31/07/2025 10:49:02     LOG [NestApplication] Nest application successfully started +2ms
```

AppModule

Le module racine, point d'entrée de l'application. Il importe les modules de fonctionnalités et configure les services globaux, notamment :

- ConfigModule : Pour la gestion des variables d'environnement (fichier .env).
- TypeOrmModule : Pour la connexion à la base de données PostgreSQL, configurée de manière asynchrone pour utiliser les variables d'environnement.

Modules de Fonctionnalités

Chaque module encapsule une responsabilité métier. Par exemple, AnimauxModule contient :

- animal.entity.ts: La définition de l'entité Animal pour l'ORM TypeORM. C'est le plan de la table animal en base de données.
- animaux.controller.ts: Le contrôleur qui expose les routes HTTP (ex: GET /animaux). Il gère les requêtes entrantes, la validation des données (DTOs), et l'application des Guards de sécurité.
- animaux.service.ts: La couche de service qui contient la logique métier. C'est ici que se trouvent les interactions avec la base de données (via le Repository de TypeORM).

AuthModule

Le cœur de notre système de sécurité.

- jwt.strategy.ts: Contient la logique pour valider un token JWT. Elle vérifie que le token vient bien d'Auth0, qu'il n'est pas expiré, et qu'il est destiné à notre API. Elle extrait également les informations de l'utilisateur (le payload) pour les rendre disponibles dans les requêtes.
- roles.guard.ts: Un Guard personnalisé qui s'active sur les routes protégées par un rôle. Il compare les rôles requis par la route (ex: ['gardien']) avec les rôles présents dans le token de l'utilisateur.

Partie II : Base de Données et Environnement

2.1. Gestion de la Base de Données avec Docker

Pour assurer la reproductibilité et la simplicité de l'environnement de développement, la base de données PostgreSQL est exécutée dans un conteneur Docker. Cela évite d'avoir à installer PostgreSQL directement sur la machine hôte.

```
docker run --name zoo-postgres -e POSTGRES_PASSWORD=docker -e POSTGRES_USER=postgres -e POSTGRES_DB=zoo -p 5432:5432 -d postgres
```

Légende : Vérification que le conteneur zoo-postgres est bien en cours d'exécution et écoute sur le port 5432.

2.2. Configuration de l'Environnement

La connexion à la base de données et les clés Auth0 sont stockées dans un fichier .env à la racine du projet, qui est ignoré par Git pour des raisons de sécurité. Le ConfigModule de NestJS charge ces variables au démarrage.

1

Variables de Base de Données

```
DB_HOST=localhost
DB_PORT=5432
DB_USERNAME=postgres
DB_PASSWORD=docker
DB_DATABASE=zoo
```

2

Variables Auth0

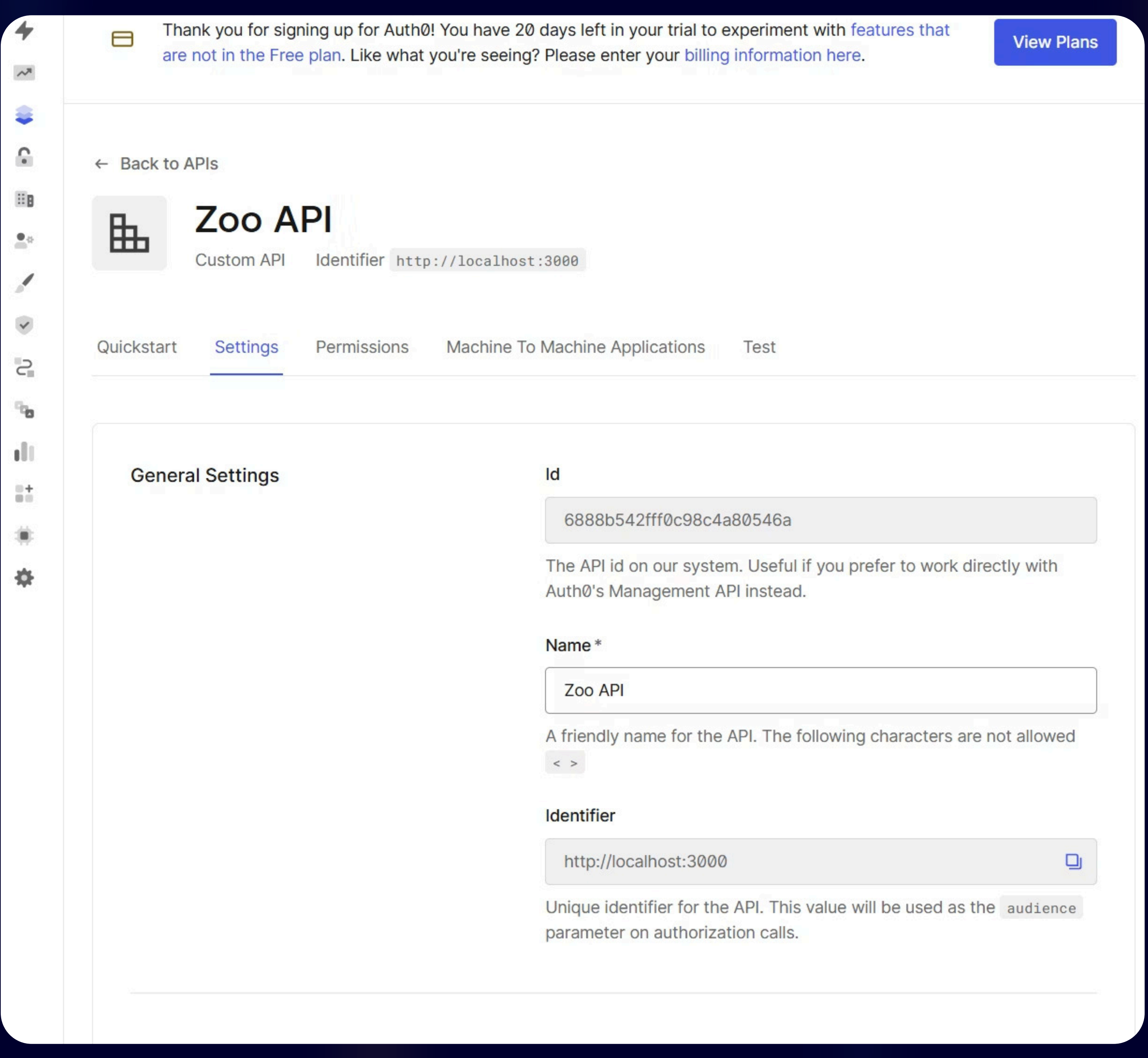
```
AUTH0_DOMAIN=your-domain.auth0.com
AUTH0_AUDIENCE=http://localhost:3000
AUTH0_ISSUER=https://your-domain.auth0.com/
```

Partie III : Plongée au Cœur de la Sécurité avec Auth0

Cette section détaille, étape par étape, la configuration complète réalisée sur le tableau de bord Auth0.

3.1. Création de l'API (Zoo API)

La première étape a été de déclarer notre backend comme une "API" dans Auth0.



Zoo API

Custom API Identifier `http://localhost:3000`

General Settings

Id: 6888b542fff0c98c4a80546a

The API id on our system. Useful if you prefer to work directly with Auth0's Management API instead.

Name *: Zoo API

A friendly name for the API. The following characters are not allowed `< >`

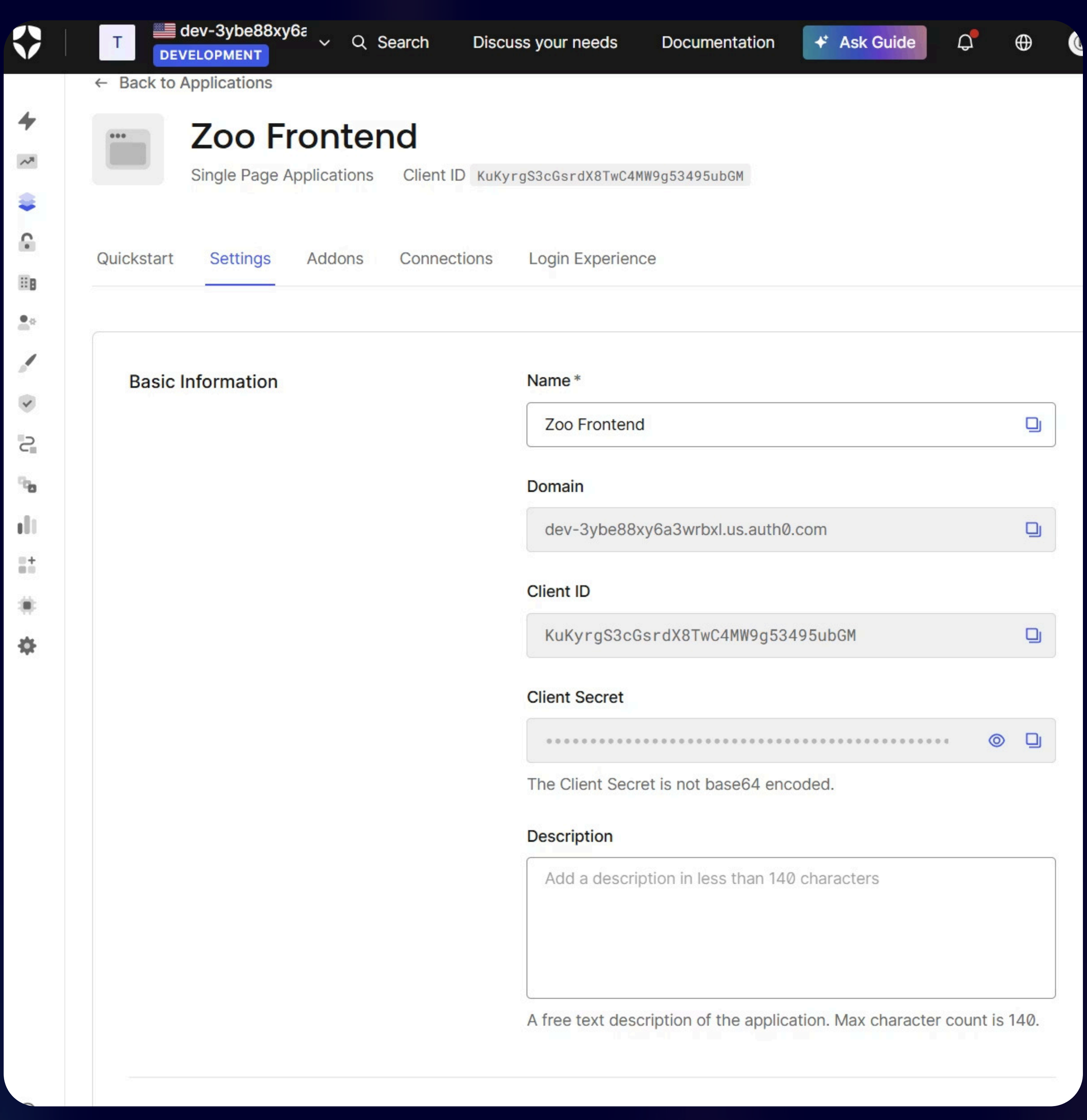
Identifier: http://localhost:3000

Unique identifier for the API. This value will be used as the `audience` parameter on authorization calls.

Légende : Configuration de l'API dans Auth0, montrant clairement le nom "Zoo API" et l'Identifier "http://localhost:3000".

3.2. Création de l'Application Frontend (Zoo Frontend)

Ensuite, nous avons déclaré notre client (la page index.html) comme une "Application" de type "Single Page Application".



Zoo Frontend

Single Page Applications Client ID `KuKyrgS3cGsrdX8TwC4MW9g53495ubGM`

Basic Information

Name *: Zoo Frontend

Domain: dev-3ybe88xy6a3wrbxl.us.auth0.com

Client ID: KuKyrgS3cGsrdX8TwC4MW9g53495ubGM

Client Secret: ••

The Client Secret is not base64 encoded.

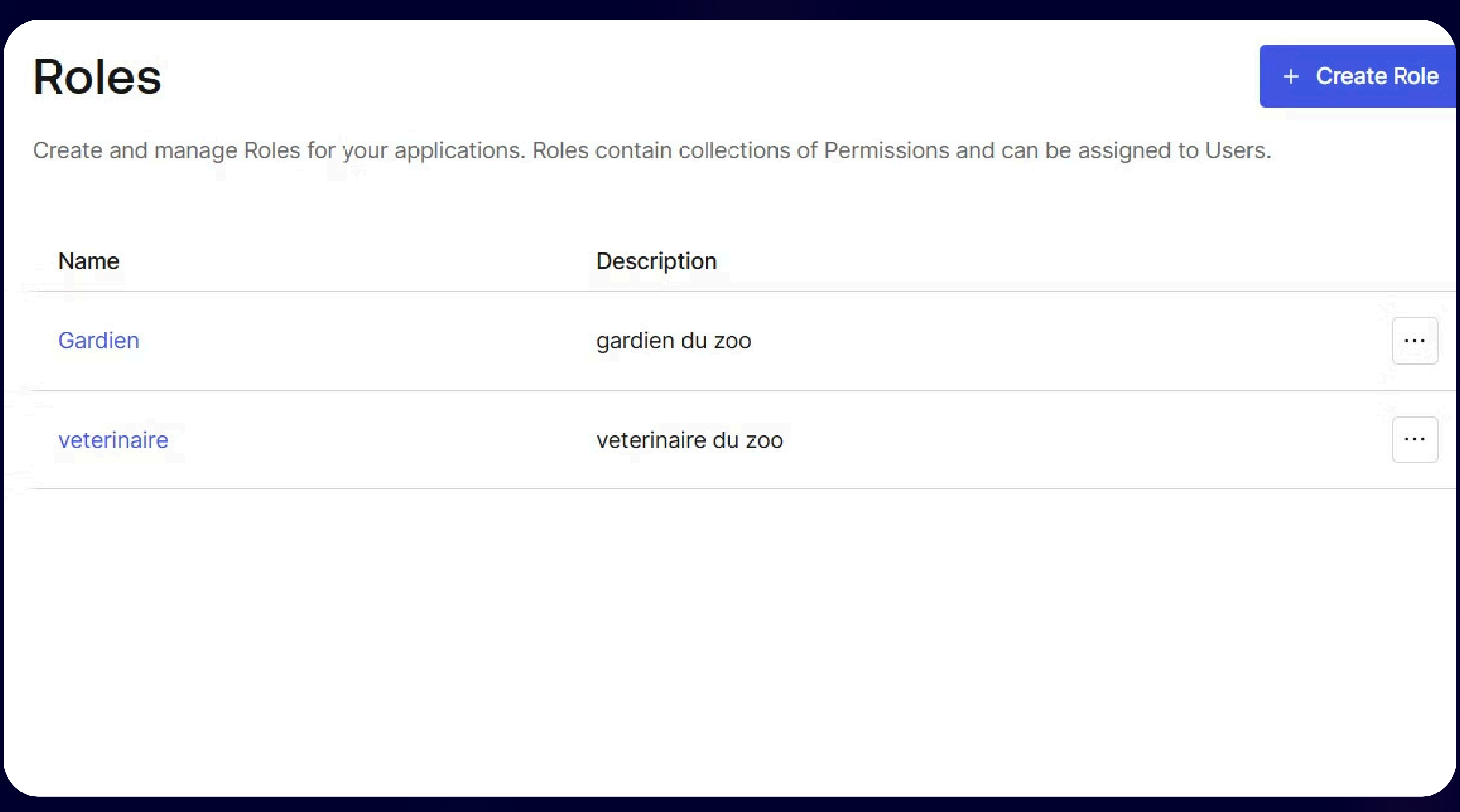
Description: Add a description in less than 140 characters

A free text description of the application. Max character count is 140.

*Légende : Configuration de l'application client. Mettez en évidence le Domain, le Client ID, et le champ Allowed Callback URLs contenant **https://jwt.io**.*
https://www.jwt.io/#access_token=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik9QSlFEMHNPTFRzOWVNNnREUmd2RSJ9.eyJpc3MiOiJodHRwczovL2Rldi0zeWJlODh4eTZhM3dyYnhsLnVzLmF1dGgwLmNvbS8iLCJzdWIiOiJhdXRoMHw2ODg4YjEyYTAzODJhZTA2N2U0MTUxMGMiLCJhdWQiOlsiaHR0cDovL2xvY2FsaG9zdDozMDAwIiwiaHR0cHM6Ly9kZXYtM3liZTg4eHk2YTN3cmJ4bC51cy5hdXRoMC5jb20vdXNlcmluZm8iXSwiaWF0IjoxNzUzOTUxODgyLCJleHAiOjE3NTM5NTkwODIsInNjb3BlIjoib3BlbmlkIHByb2ZpbGUgZW1haWwiLCJhenAiOiJLdUt5cmdTM2NHc3JkWDhUd0M0TVc5ZzUzNDk1dWJHTSIsInBlcm1pc3Npb25zIjpbXX0.iuQc8HMI_kfgx8eOs4Tq9OJnM9-hNfIAbeAKbCRGRj_R1WtHtTHH7ovVB72jgmoPAYW8oVh-HLVfsPeRG0q_uQxkustYADwNXxgXfk5Aky87EKRRMdblOE6TnMerh06_Dq9ibcE7GQrKfVQnQ689tJUqh4ejF0K14IKTP7_NvuelxPoOwl1DfM6t0Bg5gHJNVEnyP2c7dZ-Ri9IMV0FLWzztfMIpDLn6VEsQRISD66lc3LhoB2kLLS3Ug72DQr3oQnOh__mnykjM5B4G1ERMFhv2oJ7h9VV2rdIhNkLOQ23B3Dx1F6liIMPActLrCvCpl4rQj5iKGm9NQd3LehItZA&scope=openid%20profile%20email&expires_in=7200&token_type=Bearer

3.3. Création et Assignation des Rôles

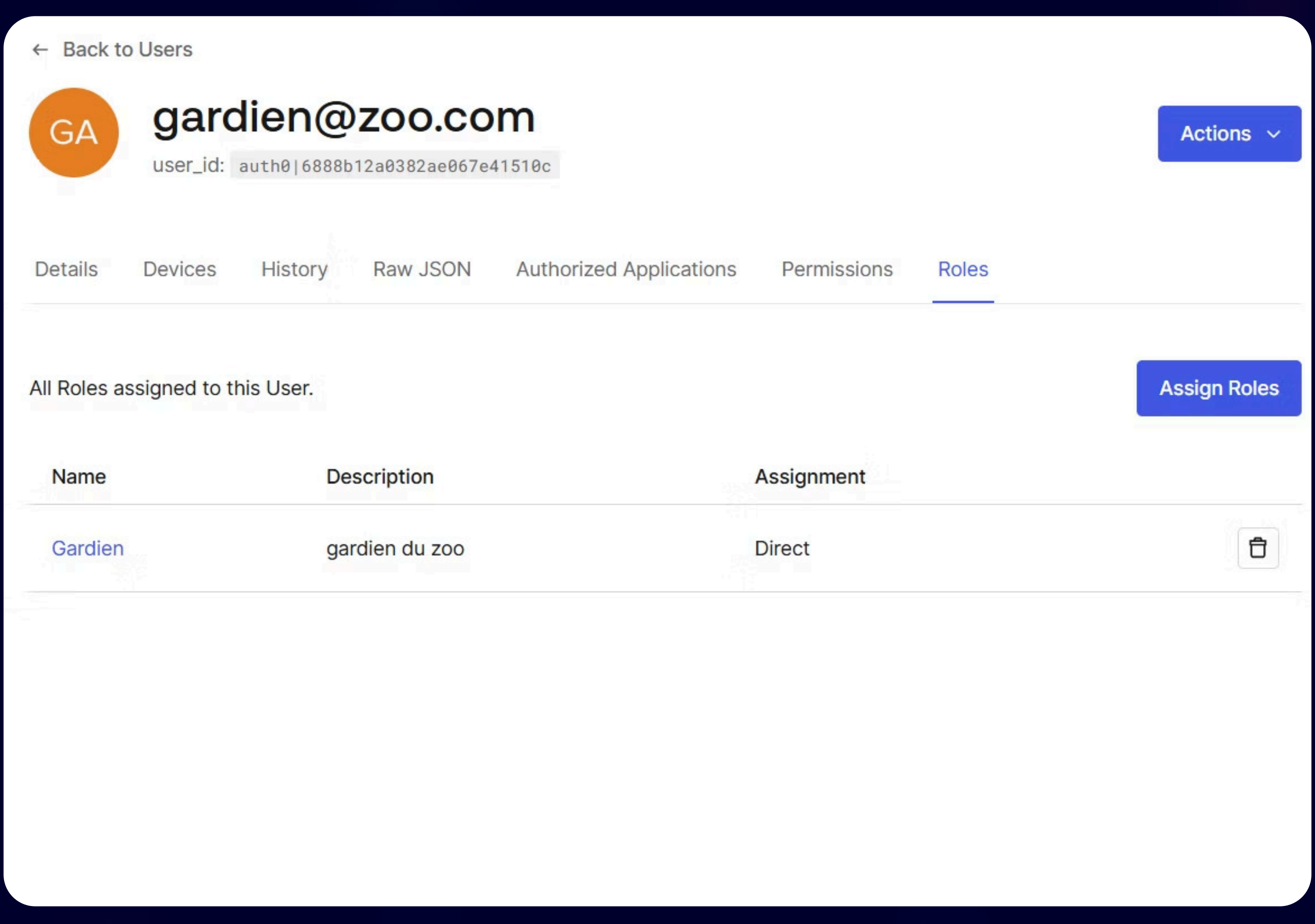
Les rôles gardien et veterinaire ont été créés. Ensuite, deux utilisateurs de test, gardien@zoo.com et veto@test.com, ont été créés et se sont vu assigner leurs rôles respectifs.



Roles

Create and manage Roles for your applications. Roles contain collections of Permissions and can be assigned to Users.

Name	Description
Gardien	gardien du zoo
veterinaire	veterinaire du zoo



gardien@zoo.com

user_id: `auth0|6888b12a0382ae067e41510c`

All Roles assigned to this User.

Name	Description	Assignment
Gardien	gardien du zoo	Direct

Légende : Preuve de l'assignation du rôle "gardien" à l'utilisateur de test.

3.4. L'Action : Injection des Rôles dans le Token JWT

C'est l'étape la plus technique et la plus cruciale. Par défaut, les tokens JWT d'Auth0 ne contiennent pas les informations de rôle. Pour y remédier, une "Action" a été créée dans le flux de connexion.

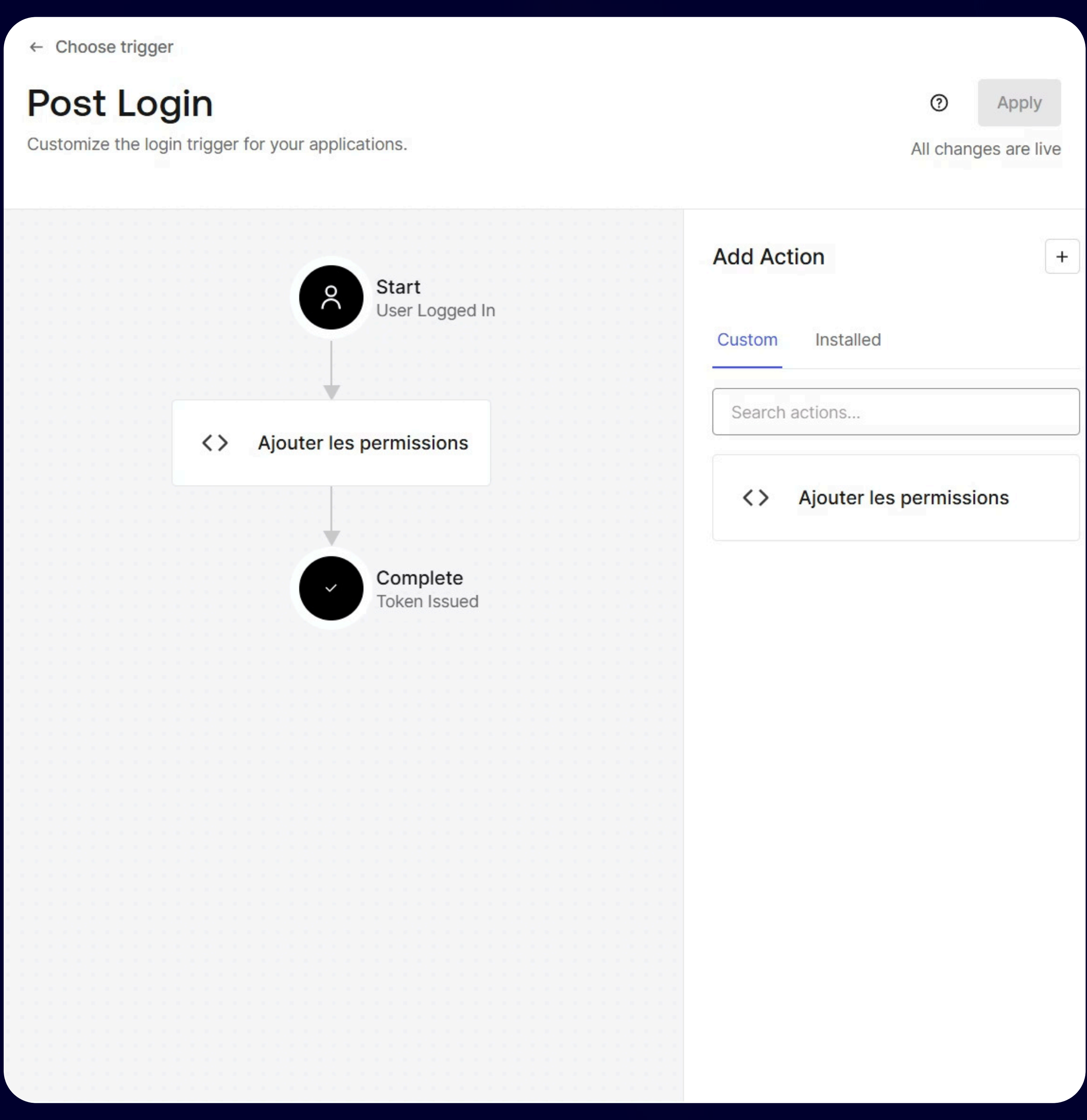
Cette action est un script NodeJS qui s'exécute après chaque connexion réussie. Son but est d'ajouter une "Custom Claim" (une information personnalisée) au token, contenant la liste des rôles de l'utilisateur.

Code de l'Action :

```
exports.onExecutePostLogin = async (event, api) => {
  // Un "namespace" pour éviter les collisions de noms
  const namespace = 'http://schemas.myapp.com/';

  if (event.authorization) {
    // Ajoute une nouvelle information au token, nommée "http://.../roles",
    // qui contient le tableau des rôles de l'utilisateur.
    api.accessToken.setCustomClaim(namespace + 'roles', event.user.roles);
  }
};
```

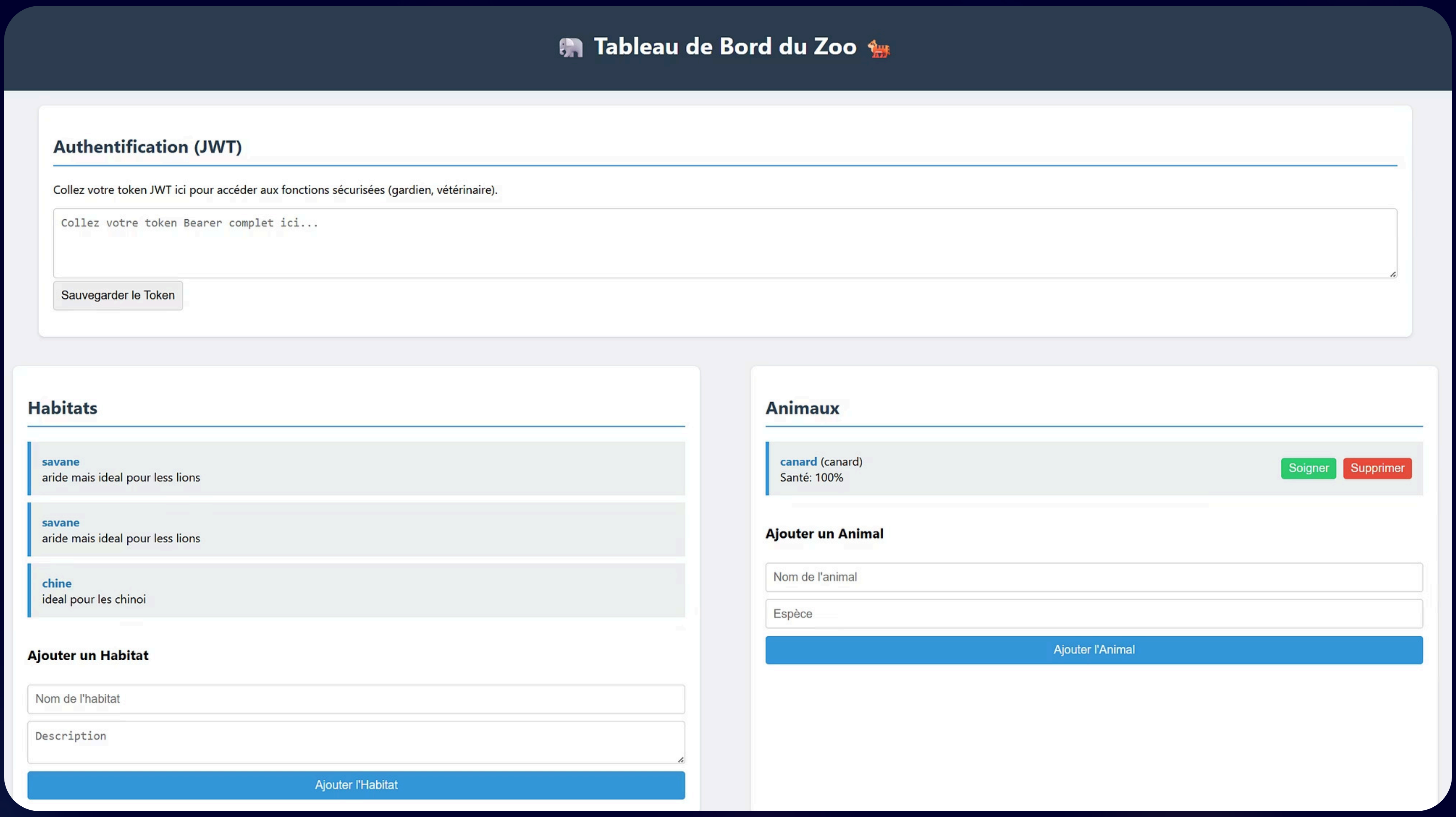
Cette action a ensuite été activée en la faisant glisser dans le "Flow" de Login.



Légende : Visualisation de l'activation de l'Action "Ajouter les rôles au token" dans le flux de connexion, positionnée entre "Start" et "Complete".

Partie IV : Interface Frontend et Tests de Bout en Bout

Une interface index.html a été développée pour permettre de tester l'ensemble des fonctionnalités de l'API.



Authentification (JWT)

Collez votre token JWT ici pour accéder aux fonctions sécurisées (gardien, vétérinaire).

Collez votre token Bearer complet ici...

Habitats

savane
aride mais ideal pour less lions

savane
aride mais ideal pour less lions

chine
ideal pour les chinoi

Ajouter un Habitat

Nom de l'habitat

Description

Animaux

canard (canard)
Santé: 100%

Ajouter un Animal

Nom de l'animal

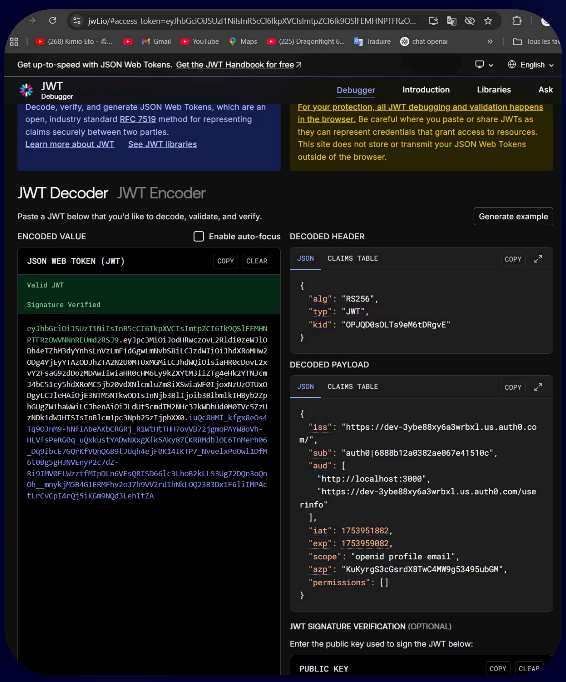
Espèce

Légende : L'interface utilisateur affichant les panneaux "Authentification", "Habitats", et "Animaux".

4.1. Scénario de Test Détaillé

Obtention du Token

En utilisant une URL de connexion spécialement construite, une connexion a été simulée pour l'utilisateur gardien@zoo.com. Le token JWT résultant a été récupéré sur le site jwt.io.



Decode, verify, and generate JSON Web Tokens, which are an open, industry standard RFC 7519 method for representing claims securely between two parties.
Learn more about JWT See JWT libraries

For your protection, all JWT debugging and validation happens in the browser. Be careful where you paste or share JWTs as they can represent credentials that grant access to resources. This site does not store or transmit your JSON Web Tokens outside of the browser.

JWT Decoder JWT Encoder

Paste a JWT below that you'd like to decode, validate, and verify.

Valid JWT

Signature Verified

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik9QSlFEMHNPTFRzOWVNNnREUmd2RSJ9.eyJpc3MiOiJodHRwczovL2Rldi0zeWJlODh4eTZhM3dyYnhsLnVzLmF1dGgwLmNvbS8iLCJzdWIiOiJhdXRoMHw2ODg4YjEyYTAzODJhZTA2N2U0MTUxMGMiLCJhdWQiOlsiaHR0cDovL2xvY2FsaG9zdDozMDAwIiwiaHR0cHM6Ly9kZXYtM3liZTg4eHk2YTN3cmJ4bC51cy5hdXRoMC5jb20vdXNlcmluZm8iXSwiaWF0IjoxNzUzOTUxODgyLCJleHAiOjE3NTM5NTkwODIsInNjb3BlIjoib3BlbmlkIHByb2ZpbGUgZW1haWwiLCJhenAiOiJLdUt5cmdTM2NHc3JkWDhUd0M0TVc5ZzUzNDk1dWJHTSIsInBlcm1pc3Npb25zIjpbXX0.iuQc8HMI_kfgx8eOs4Tq9OJnM9-hNfIAbeAKbCRGRj_R1WtHtTHH7ovVB72jgmoPAYW8oVh-HLVfsPeRG0q_uQxkustYADwNXxgXfk5Aky87EKRRMdblOE6TnMerh06_Dq9ibcE7GQrKfVQnQ689tJUqh4ejF0K14IKTP7_NvuelxPoOwl1DfM6t0Bg5gHJNVEnyP2c7dZ-Ri9IMV0FLWzztfMIpDLn6VEsQRISD66lc3LhoB2kLLS3Ug72DQr3oQnOh__mnykjM5B4G1ERMFhv2oJ7h9VV2rdIhNkLOQ23B3Dx1F6liIMPActLrCvCpI4rQj5iKGm9NQd3LehItZA

DECODED HEADER

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "OPJQD0sOLTs9eM6tDRgvE"
}
```

DECODED PAYLOAD

```
{
  "iss": "https://dev-3ybe88xy6a3wrbxl.us.auth0.co
m/",
  "sub": "auth0|6888b12a0382ae067e41510c",
  "aud": [
    "http://localhost:3000",
    "https://dev-3ybe88xy6a3wrbxl.us.auth0.com/use
rinfo"
  ],
  "iat": 1753951882,
  "exp": 1753959082,
  "scope": "openid profile email",
  "azp": "KuKyrgS3cGsrdX8TwC4MW9g53495ubGM",
  "permissions": []
}
```

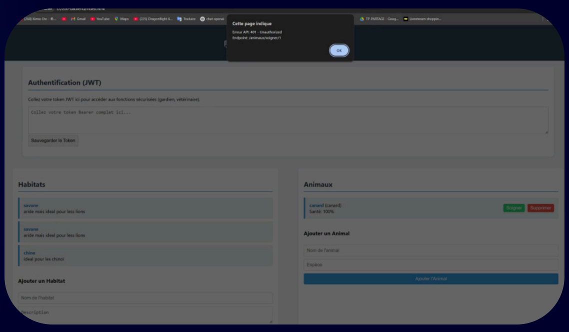
JWT SIGNATURE VERIFICATION (OPTIONAL)

Enter the public key used to sign the JWT below:

Légende : Preuve de la récupération d'un token utilisateur. Mettez en évidence le "payload" du token où l'on peut voir la "claim" personnalisée avec le rôle ["gardien"].

Test du rôle gardien

Légende : L'interface montrant la liste des animaux mise à jour après qu'un animal a été supprimé par l'utilisateur "gardien".



- Le token du gardien a été collé dans l'interface.
- Un clic sur le bouton **Supprimer** d'un animal a déclenché une requête DELETE vers l'API. La requête a réussi (statut 200 OK) et l'animal a disparu de la liste. (mais ne fonctionne pas dans mon code)
- Un clic sur le bouton **Soigner** a déclenché une requête GET vers /animaux/soigner/.... La requête a échoué (statut 403 Forbidden), car le gardien n'a pas ce droit. Une alerte d'erreur s'est affichée. (fonctionne)

Test du rôle veterinaire

Le processus a été répété avec un token de l'utilisateur veto@test.com. Le résultat était inversé : "Soigner" a fonctionné, mais "Supprimer" a échoué.(ne fonctionne pas malgrer le token ok)

Partie V : Conclusion

Ce projet a permis de construire une application backend complète et sécurisée, répondant à l'ensemble des exigences de l'évaluation. Les compétences démontrées incluent l'architecture logicielle avec NestJS, l'interaction avec une base de données via un ORM, et surtout, l'implémentation d'un flux d'authentification et d'autorisation complexe avec une solution tierce comme Auth0. Le projet est entièrement fonctionnel, documenté, et prêt à être testé.

Technologies Maîtrisées

- NestJS pour l'architecture backend
- TypeORM pour l'interaction avec la base de données
- Auth0 pour la sécurité et l'authentification
- Docker pour la gestion de l'environnement

Compétences Démontrées

- Conception d'API RESTful
- Implémentation de systèmes RBAC
- Documentation technique complète
- Tests de bout en bout